Stack type snapshot buffer handles nested interrupts

Title Patent Application: A data processor with a functional unit, a register file, a data memory and a snapshot buffer which during interrupt handling saves processor state informations in a snapshot buffer, and at a subsequent interrupt during handling of an actual interrupt saves the snapshot buffer content in a data memory facility with a multibit access port facility, a data processing facility with such data processor embedded, and a method for operating such data processor that is arranged for handling nested interrupts.

BACKGROUND OF THE INVENTION

The invention relates to a data processor comprising one or more functional units, one or more register files, a data memory, and moreover a snapshot buffer that accommodates to store state informations of the processor during an interrupt condition in respective buffer elements as has furthermore been recited in the preamble of Claim 1. For being able to obtain high-quality operation schedules in VLIW and other state of the art processors, ideally all hardware characteristics that are relevant to the compiler should be visible to the compiler. Although not being limited thereto, a particular relevant category of such informations pertains to the various latencies of respectively associated operations. High performance pipelined processors use latency values that are in excess of one, to thereby raise performance. In operation, the latency values should be exactly specified. This implies that not only the total duration of the operation will be specified, but also the precise instants should be known at which specific operations will occupy specific hardware for the consuming of operands, the processing of data, and the production of results. Thereby, superior schedules with minimum register pressure can be obtained.

The above scheduling approach is called NUAL-EQ semantics (Non Unit Assumed Latency with Equal). Always interruptible programmable processors that support compile-time scheduling based on NUAL-EQ semantics require that an exact snapshot be taken from the internal processor pipeline that should be restored as part of the interrupt handling scheme. In fact, the procedure that has been more commonly in use, and which flushes the processor pipeline state could subsequently lead to an incorrect behaviour of the processor. Such a snapshot will be stored in a snapshot buffer as has been disclosed in

published PCT Patent application WO 02/33570 A2, assigned to the present assignee and co-invented by the present inventor, and being herein incorporated by reference.

Now, the present inventor has duly recognized the relevance of so-called ***nested interrupts***, wherein the handling of a previous interrupt condition has not been completed when a subsequent interrupt will occur. In such case there must be a possibility to save and restore multiple and simultaneous snapshots. Although there would in principle exist various different approaches to solve this problem, the least expensive one is to allow the snapshot buffer to be written and read by a load/store controller unit, in such way that the latest snapshot that had been taken at the start of the handling of the current interrupt, can be stored in a background memory to make place for a new snapshot that is taken when a new interrupt materializes.

Now, the snapshot buffer can be made to behave like a register file in order to obtain the above indicated functionality without requiring a change in the instruction set architecture (ISA) of the processor. However, the adding of the snapshot buffer to the register file address space of the processor implies that additional register addressing bits are required in the instruction encoding. Therefore, a method is needed to reduce or even fully eliminate the addressing overhead, by carefully implementing the snapshot buffer and incorporating this buffer in the processor architecture.

This particular problem has not been approached by the reference cited hereabove. There, the saving of address bits has been attained by using serial ***scan chains***. These serial scan chains would allow to shift snapshot data into or out of the snapshot buffer without requiring explicit addressing of register locations within the snapshot buffer. However, a distinct disadvantage of the use of scan chains is that much wiring is necessary to effectively constitute the chain from all of the various buffer flipflops. Finally, the shifting required to read data from or to write data into the snapshot buffer leads to unnecessary switching activity and thereby to increased power dissipation.

The inventor has recognized the advantageous feature of providing the interruptible data processor with a snapshot buffer composed of shadow flipflops. Each shadow flipflop has a corresponding normal flipflop in one of the resources of which the state must be saved and restored by the processor hardware before and after the interrupt handling, such in correspondence with the above cited reference.

Now, it is furthermore proposed to have the snapshot buffer behave like a ***stack-based*** register file that can be read or written by a standard load/store unit. Since the snapshot buffer behaves like a stack, no register index is required to address it. Furthermore,

no shifting of snapshot data is required, as would have been the case with the scan chain based solution of the reference. Instead, a stack pointer is kept internally in the snapshot buffer. Reading from the snapshot buffer, to store a snapshot into background memory, pops the top of the stack and decrements the stack pointer value so that it will point to the previous top of the stack. Writing to the snapshot buffer, to reload a snapshot again from background memory, will push the new data onto the stack and increment the stack pointer so that it will point to the next top of the stack. Note that writing and reading in the snapshot buffer in this manner is only required when interrupts are actually nested. Otherwise, a conventional procedure would suffice.

SUMMARY TO THE INVENTION

In consequence, amongst other things, it is an object of the present invention to allow the occurrence of multiple and nested interrupts by having a secondary data storage level care for the saving of all snapshots except for the most recent one.

Now therefore, according to one of its aspects the invention is characterized according to the characterizing part of Claim 1.

The invention also relates to a data processing facility with such data processor embedded, and to a method for operating a data processor arranged for handling nested interrupts in the above indicated manner. Further advantageous aspects of the invention are recited in dependent Claims.

BRIEF DESCRIPTION OF THE DRAWING

These and further aspects and advantages of the invention will be discussed more in detail hereinafter with reference to the disclosure of preferred embodiments, and in particular with reference to the appended Figures that show:

Figure 1, a block diagram of a VLIW processor with a stack-based snapshot buffer;

Figure 2, a block diagram of a stack-based snapshot buffer;

Figure 3, a shadow flipflop interconnected to a normal resource flipflop.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Figure 1 illustrates a block diagram embodiment of a VLIW processor with a stack-based snapshot buffer. In the Figure, the number of interconnections actally shown has been kept as low as possible for thereby getting a clear Figure whilst actually indicating only

those interconnections that were considered necessary to explain the functioning of the structure. Now, the arrangement contains two register files (RF_0, RF_1) 22, 24, four issue slots (UC_0, UC_1, UC_2, UC_3) 32, 34, 36, 38, an interconnection network (CN) 28 interconnecting the register files to the issue slots, and a controller (SQ) 26. The first issue slot (UC_0) 32 is the only issue slot actually used during interrupt handling. During such interrupt handling, the various states of relevant resources, together with the relevant state of the sequencer will be copied into the shadow flipflops of the snapshot buffer (SS) 20. The latter is exclusively connected to a load/store unit (LSU) 30 located within UC_0 32 and operating as an additional register file to the above register files RF_0 22 and RF_1 24. Like any conventional load/store unit, the element LSI 30 has access to a background data memory (DM) 40 to therein store and therefrom load snapshot data in the case of handling nested interrupts.

Figure 2 illustrates a block diagram of the internal arrangement of a stack-based snapshot buffer. The buffer is composed of a plurality of shadow flipflops not shown separately that have been organized in a set 50 of parallel words of which only word 52 has been indicated. Word length dimensioning is done according to need and available processor facilities such as data path width. Each shadow flipflop is connected to a corresponding state indicating flipflop that is located in a processor resource. During the handling of an interrupt, this state must be saved, cf. Figure 3 hereinafter. In the snapshot buffer, the inputs to the various shadow words are connected to a demultiplexor 54 for allowing at any time the writing of exactly one word. Likewise, in the snapshot buffer, the outputs from the various shadow words are fed to a multiplexor 56 for allowing at any time the reading of exactly one word.

Both the demultiplexor 54 and the multiplexor 56 are controlled by a stack pointer from a stack pointer register 58 that is located in the snapshot buffer as well. In fact, reading from the snapshot buffer on line 62 is only required at the start of a nested interrupt, and writing to the snapshot buffer from line 60 is only required at the end of a nested interrupt. Therefore, these two operations will never occur simultaneously and a single pointer register 58 could be sufficient to control the interleaved addressing of different words in the stack buffer. The pointer value is retrocoupled on line 62 to pointer update control 66 for subsequent reloading of pointer register 58. As shown, three-operation control line 68 will allow respective read, write, and no-operation modes with respect to the pointer value. The effective operations are executed in decrementing element (-1), incrementing element (+1), and no-operation element (through a straightaway retrocoupling).

Because in the embodiment the snapshot buffer will maintain its own internal stack pointer, none of the above read/write commands will require a register address, as would have been required for standard random access register files. Hence, no additional instruction bits will be required for addressing the registers in the snapshot buffer. Note that the actual ***value*** of the stack pointer is not related to the ***level*** of an interrupt, but instead to the sequence according to which for a particular interrupt the data will be written to the status registers. As soon as a subsequent interrupt will become manifest, the contents of the set of shadow registers will be written to the stack. As soon as the next interrupt arrives, the shadow registers are copied to DM 40, to make place for newer data.

Figure 3 illustrates a shadow flipflop 72 interconnected to a normal resource flipflop 70, cf the published reference Patent Application WO 02/33570 A2 cited supra. The interrupted line (a) at left delimits the standard operational hardware with standard or original flipflop 70 and input gate facility 74. The input gate facility 74 will receive consume data 78 and the flipflop 70 will output process data 82. The shadow flipflop 72 outputs save data on line 84. The shadow flipflop input is fed by save/store multiplexor 76 that is controlled by control signal 86. The two halves of the Figure are mutually cross-coupled as shown.

At the start of each interrupt handling, a complete snapshot is taken from the relevant processor state within a single clock cycle through instantly copying the value of each normal flipflop to its corresponding shadow flipflop. Similarly, at the end of each interrupt handling, the complete snapshot is restored from the snapshot buffer to the processor proper by instantly copying each shadow flipflop to its corresponding normal flipflop.

Before enabling the nesting of an interrupt, the contents of the snapshot buffer can be stored in data memory through using the load/store unit popping of snapshot data organized in words from the stack. This feature will then free the snapshot buffer to take a new snapshot at the start of the handling of each such nested interrupt. Upon return from a nested interrupt, after restoring the current snapshot buffer contents, the original snapshot data from an interrupted service routine can be loaded from data memory and be written to the snapshot buffer through using a load/store unit that pushes snapshot data organized in words onto the stack.

Low power can be obtained by deactivating the snapshot buffer as much as feasible. This is done in the following manner. All shadow flipflops in the snapshot buffer are only clocked during the actual taking of a snapshot, which may imply activating the snapshot buffer during only a single clock cycle. Furthermore, only the shadow flipflops pointed to by

the stack pointer as the top-of-stack-plus-one are clocked during a stack push operation. Finally, the stack pointer itself is only clocked during stack pointer updates that are caused by the popping and pushing, respectively, of the snapshot buffer stack.

An advantageous field of application of the invention are embedded digital signal processors (**DSP**). The invention will be further applicable in various interruptible embedded processor architectures that employ scheduling based on **NUAL-EQ**, and that would therefore require pipeline snapshots instead of pipeline flushing. Specific fields of application would be video processing, video codecs, audio codecs, audio graphics, 3G telecom, Voice-over-packet, and many others.